Universitat Oberta de Catalunya

uoc.edu

Save The SpaceShip!

David García Núñez
Grado en Ingeniería Informática
Videojuegos

Prof. Consultor: **Jordi Duch Gavaldà**
Prof. Consultor: **Helio Tejedor Navarro**
Prof: **Joan Arnedo Moreno**

06/01/2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Save the Spaceship!*
Nombre del autor:	*David García Núñez*
Nombre del consultor/a:	*Jordi Duch Gavaldà* *Helio Tejedor Navarro*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	01/2019
Titulación:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Videojuegos*
Idioma del trabajo:	*Castellano*
Palabras clave	*Videojuego, arcade, retro*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Este proyecto contiene dos objetivos muy claros: en primer lugar, la producción de un videojuego, desde el comienzo, cuando no es más que una idea en la cabeza de un estudiante, hasta la entrega de un producto final terminado. En segundo lugar, y el que más valía posee, es el aprendizaje que conlleva ejecutarlo y apoyarse en todo aquello que se ha aprendido en durante la carrera y que puede vincularse al proyecto.

El contexto del producto acabado es el mero placer lúdico, pero el contexto del proceso es la acumulación de una riqueza técnica, de gestión de proyecto y sobre todo de aprendizaje de errores y como tratar con ellos.

La metodología empleada es simple: idear, planificar, ejecutar, revisar y corregir. Se ha intentado la adhesión a la planificación dada al comienzo, aunque como en todo proyecto, han ido apareciendo obstáculos imprevistos y desviaciones que han sido tratados consecuentemente; es parte del aprendizaje, un resultado más junto con el producto final acabado.

La conclusión a más alto nivel de este proyecto es que debemos ser más humildes en nuestras metas, pero solo lo justo y necesario para dejar sobresalir un poco nuestra ambición, pues ese es el impulso que nos saca del confort y nos hace avanzar a lugares inexplorados.

Abstract (in English, 250 words or less):

This project contains two very clear objectives: first, the production of a a videogame, from the beginning, when it is nothing more than an idea in the mind of a student, until the delivery of a finished product. Secondly, and the one that has the most value, is the learning that entails executing it and relying on everything that has been learned during the career and that can be linked to the project.

The context of the finished product is the mere playful pleasure, but the context of the process is the accumulation of technical wealth, project management and above all, learning from the mistakes and how to deal with them.

The methodology used is simple: devise, plan, execute, revise and correct. Attempts have been made to adhere to the planning given at the beginning, although, as in any project, unforeseen obstacles and deviations have arisen that have been treated accordingly; It is part of the learning process together with the finished product.

The conclusion at the highest level of this project is that we should be humbler in our goals, but only just the necessary to let our ambition stand out a bit, because that is the impulse that takes us out of comfort and makes us advance to unexplored places.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Bajo una capa lúdica, se esconde un intrincado mecanismo que aúna múltiples facetas o ramas de la informática y de su inmediata progenitora, las matemáticas. Un videojuego exige dominar técnicas tan variadas como la simulación de las fuerzas físicas, interacción entre entes abstractos, paso de mensajes, sincronización, generación de números pseudo-aleatorios, manipulación gráfica, gestión de eventos, etc.

Además de lo descrito en el párrafo anterior, al autor de un videojuego se le exige (en el caso de proyectos en solitario): cierto dominio en la composición de variado arte gráfico, efectos de sonido y música, además de una trama literaria que permita vertebrar una historia. A fin de cuentas, el creador de un videojuego ha de levantar un mundo al completo, que sostenga su creación y esta sea percibida como completa por el usuario que se adentre en ese universo: la partida.

Curioso nombre: “la partida”. Aunque a primera oída nos remita al mundo deportivo (el cual los videojuegos cada vez más le comen terreno a la acepción), entraña otras lecturas más emocionantes si cabe. “El héroe parte de su origen a cumplir con su destino”. Cuando esa moneda de 25 pesetas era arrojada a la boca de la recreativa, esta nos abre paso a ese universo sintético por un tiempo limitado y definido por nuestra pericia. Como un Caronte digital, que nos aleje del mundo natural al de los artificios...aunque afortunadamente para nosotros, nos devuelva a la orilla sanos y salvos ocurra lo que ocurra.

Este proyecto quiere andar parte de la senda que recorren los creadores de videojuegos originales; aquellos que trabajaban en solitario para crear nuevos universos sencillos pero desafiantes: **Crear un arcade a la vieja usanza que permita desafiar a los jugadores**.

En cuanto a la relevancia de este reto, es notoria la tendencia hoy en día de revivir aquellos videojuegos que, especialmente en las décadas 80 y 90, dieron paso a las grandes obras, conocidas como triple-A; superproducciones lejos de los limitadísimos recursos de un único desarrollador. **La vuelta de los videojuegos retro, que han llegado para crear, de nuevo, un genero propio que demanda su espacio**.

Este genero permite a los creadores trabajar en solitario o en equipos muy reducidos y con escasos recursos, tanto económicos como la nueva moneda

de este siglo: el tiempo. Afortunadamente, y gracias a Internet, podemos compartir estos recursos, asociarnos, encontrar puntos comunes y crear. Sobre todo, esto último, ejercitar el derecho pleno a crear.

Este proyecto quiere aunar todo lo comentado: el proceso y el producto. **El autor quiere que este viaje sirva de propósito múltiple: Crear un videojuego retro para poner a prueba y ensayo lo aprendido en la carrera, aprender como llevar a cabo una empresa de esta magnitud y cerrar el círculo con una producción terminada**.

1.2. Objetivos del Trabajo

- Elaborar una idea inicial de concepto para un videojuego de estilo retro.
- Desgranar la idea en un plan de proyecto, acorde en el tiempo a las entregas pautadas de las distintas PEC.
- Crear arte para el videojuego y en su caso, realizar búsqueda de arte con licencia libre para incorporarlo.
- Desarrollar el videojuego teniendo muy en cuenta los conocimientos adquiridos durante la carrera.
- Finalizar el proyecto con un producto redondo. Esto es, finalizado correctamente y funcional.
- Documentar las distintas fases y plasmar su desarrollo en esta memoria y video adjunto.
- Aprender de los errores y discutir posibles mejoras.

1.3. Enfoque y método seguido

Desde el principio se tuvo muy claro el enfoque del proyecto: **un juego de estilo retro**. Un arcade que rememorase a aquellos que poblaban las recreativas de barrio de las décadas de los 80 y 90.

Dado que uno de los objetivos es “aprender” el proceso de diseño y construcción desde el principio, **se optó por crear un producto desde cero** y no una obra derivada que mejorase o demostrase una alternativa a un producto ya establecido. En el sentido de “estilo de videojuego”, sí que se inspira parcialmente en ciertos títulos publicados, como se verá en el capítulo 2.

Tal y como los propios profesores expresaron al comienzo del curso, **el tiempo iba a ser un elemento clave en el proyecto**. En el sentido de que nos va a limitar el alcance y ambición de las características del juego. Es decir, un proyecto demasiado ambicioso y lejos de las metas en las que se debe circunscribir el proyecto podría avocar a este al fracaso (producto incompleto).

Con esta limitación, **existía una necesidad de ahorrar "unidades" de tiempo**, que llevó a maximizar el uso de herramientas integradas de desarrollo y recursos con una licencia adecuada para centrarnos en las tareas más cercanas a los objetivos. Por ejemplo, se ha usado el entorno Game Maker 2 (edición Escritorio) para el desarrollo y arte con licencia libre para algunos recursos.

La estrategia ideal hubiera sido elaborar un proyecto sin el uso de *middleware*. Es decir, elegir un lenguaje cercano al sistema (C o C++), un editor de texto (Vim, Emacs, etc.) sencillo y un editor de sprites (Aseprite, etc.), al estilo del proyecto handmade hero. O como mucho, con el apoyo limitado de una librería multimedia (por ejemplo, SDL) y hacer que las propias restricciones guíen el proceso.

No obstante, eso requeriría una inversión de tiempo desproporcionada, inabarcable desde la perspectiva de un trabajo final de grado. Por lo que **se ha de optar por la mediación de un entorno de desarrollo apropiado a la creación de videojuegos que nos permitiese abstraernos de las capas de nivel más bajo**; por ejemplo, control de la memoria, carga de recursos, etc.

Por disponer de una licencia comercial válida, **se optó por desarrollar el videojuego con Game Maker 2**. Dicho entorno posee todos los mecanismos necesarios para desarrollar videojuegos en 2D, por lo que, en principio, resultó apropiado para el proyecto.

1.4. Planificación del Trabajo

La planificación a grandes rasgos viene muy demarcada por las entregas programadas para la asignatura: PEC 1, PEC 2, y PEC 3 o Documento de diseño del videojuego (en adelante, DDV), versión parcial, versión final y entrega final. Además, se ajusta a lo ya descrito en el DDV:

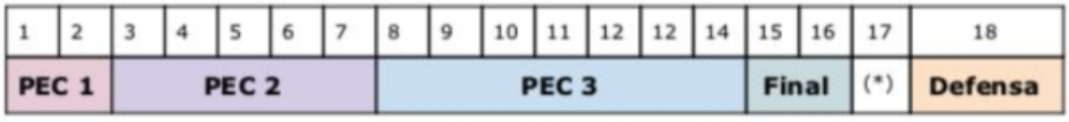


Ilustración 1 distribución por semanas

A continuación, se ilustra el diagrama de Gantt usado, el cual comprende las distintas etapas planificadas.

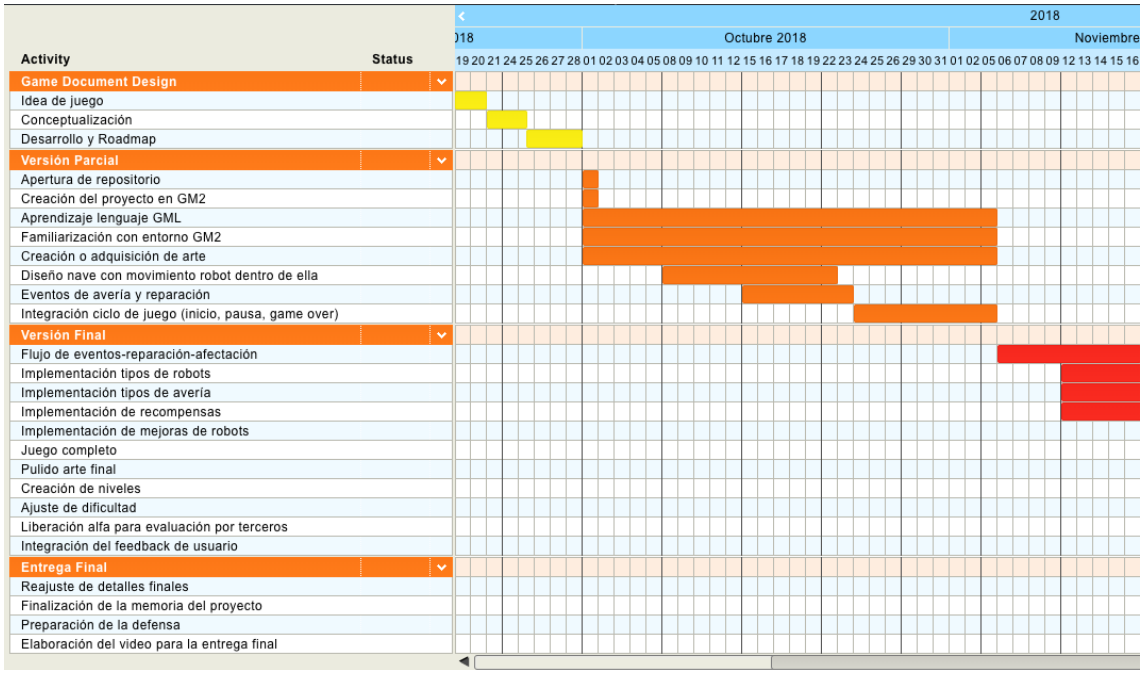


Ilustración 2 diagrama de Gantt 1/2

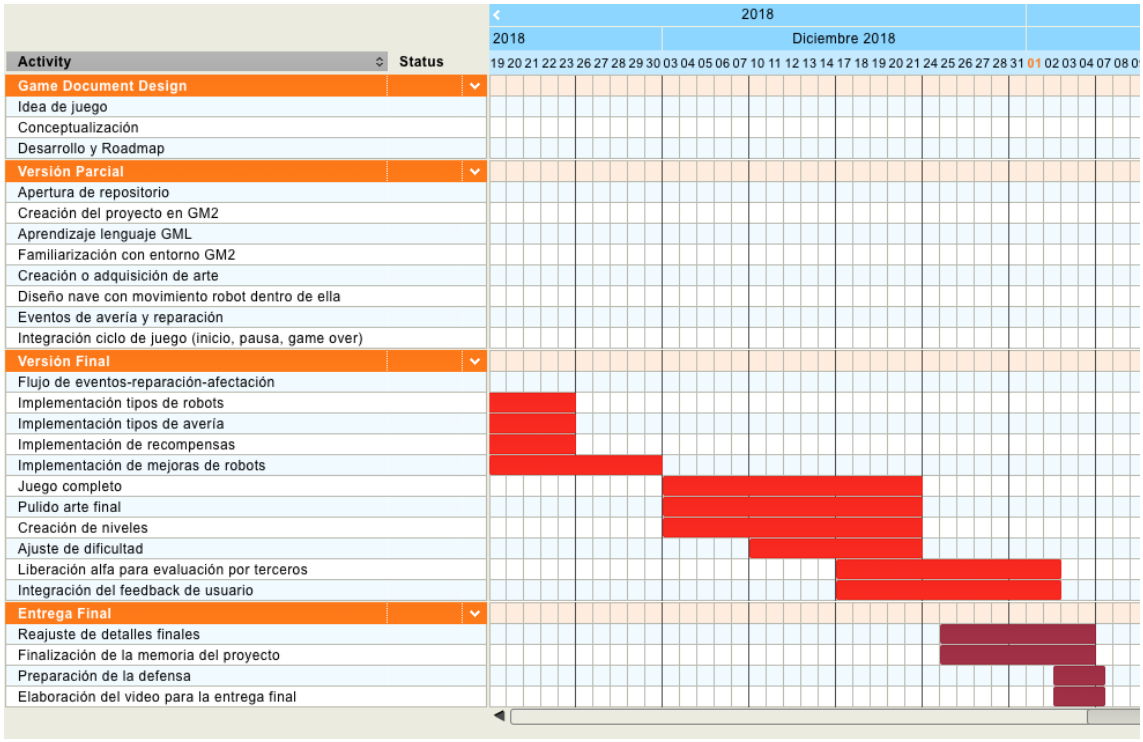


Ilustración 3 diagrama de Gantt 2/2

Como toda planificación, han existido etapas que se han extendido mucho más de lo planificado como, por ejemplo, el aprendizaje de *Game Maker Lenguage*, el entorno de desarrollo y la librería de funciones (extensa) de GM2. Este aprendizaje se entrelazó con otras etapas e incluso durante las etapas finales de la PEC 3, se descubrían nuevas formas de elaborar tareas, funciones más adecuadas o patrones comunes entre desarrolladores de GM2.

Se comenta ya aquí, que la programación se efectuó de forma lineal, sin embargo, como se verá en las conclusiones, un videojuego, como cualquier obra de arte, es una criatura viva que va abriendo a su creador nuevas posibilidades a explorar mientras va cerrando otras que parecían a este último más adecuadas al comienzo del proyecto.

Además, ciertas etapas son iterativas. En especial, el balanceo de la dificultad, el arte, etc. La propia inclusión de un elemento nuevo, tal como un tipo de evento, provoca una nueva iteración sobre fases o etapas ya consumidas.

1.5. Breve sumario de productos obtenidos

El resultado es un videojuego completo, con su ciclo de inicio, pausa, consulta de instrucciones de juego y pantalla de *game over*. La dificultad es incremental e infinita; luego, como consecuencia, el juego se vuelve cuasi imposible de terminar debido a los incrementos de dificultad.

Esto último no es accidental, los arcades antiguos eran diseñados en este sentido para animar a los jugadores a superarse y batir el record establecido. Por ejemplo, el juego del Tetris original.

El videojuego se encuentra disponible en forma de ejecutable para los sistemas Microsoft Windows.

Además, se consideran productos finales esta misma memoria, así como el video final de descripción del desarrollo.

Más importante, y como se describe en el resumen inicial de la memoria, el mejor producto ha sido el proceso en sí. Experiencia enriquecedora y muy necesaria si se desea acometer este tipo de proyectos en el ámbito profesional o con un mínimo de calidad en el mundo amateur.

1.6. Breve descripción de los otros capítulos de la memoria

- Capitulo 2: Describe el estado del arte del genero escogido y las decisiones sobre las herramientas empleadas finalmente, sus alternativas y una justificación de la decisión tomada.
- Capitulo 3: Define el juego, la referencia a otros títulos similares, su funcionamiento, interacción del jugador con este y una revisión de la tecnología empleada.
- Capitulo 4: Se explica el contexto del juego respecto a su historia, personajes, objetivos que debe cumplir el jugador además de arte conceptual de cuando se comenzó a bosquejar la idea.
- Capitulo 5: Se describe el juego desde el nivel técnico, la justificación del uso de ciertas herramientas y sus requisitos. También se describen los recursos gráficos y sonoros empleados en el juego.
- Capitulo 6: Inclusión de diversas experiencias de usuario en el programa de inclusión de feedback durante el desarrollo.
- Capitulo 7: Conclusiones generales del proyecto.
- Capitulo 8: Glosario con ciertos términos usados en la memoria.
- Capitulo 9: Bibliografía, Webgrafía y otros recursos usados.
- Capitulo 10: Anexos con los requerimientos, instrucciones y repositorio del código fuente del videojuego.

2. Estado del arte

2.1. En el género escogido: arcade con componentes retro

Como se ha comentado anteriormente, el estilo de las recreativas de los 80 y 90 se vuelve a imponer con fuerza en estos tiempos. El estilo arcade permite a desarrolladores indie y aficionados crear videojuegos con un esfuerzo relativamente menor. Sobre todo, particularmente, en equipos muy reducidos o incluso compuestos por una sola persona encargada de todos los bloques (animación, sprites, música, programación, etc)

Ilustración 4 Máquinas arcade de los 80 y 90

Este tipo de juego era especialmente adecuado para las máquinas de entonces, con ciertas limitaciones de hardware. Además, creaban un ambiente competitivo, caracterizado por el intento de batir el record que la máquina poseía. Esto también redundaba en un beneficio económico para el creador y el local donde estaba instalada, ya que las partidas eran pagadas con una moneda de 25 pesetas de la época.

El nuevo renacimiento del género comenzó con la apertura al público en general del desarrollo de videojuegos, en parte gracias a programas como Game Maker o Unity. Con este tipo de software, el creador o creadores ya no necesitaban ser unos expertos en el lenguaje o aprender a apurar cada ciclo del procesador para que el juego no padeciese de ralentización o directamente ni tan siquiera pudiese caber en memoria.

Otro punto crucial, podemos observarlo en los primeros juegos para dispositivos móviles. Aunque actualmente estos dispositivos poseen una potencia inimaginable hace unos años, en sus primeras versiones los juegos publicados debían ser sencillos. Esto hacía que los arcades clásicos fueran un género perfecto para esa etapa.

Tampoco debemos menospreciar el revival que se está viviendo con los emuladores de máquinas antiguas. Cómo, por ejemplo, la videoconsola Super Nintendo en su versión mini. Un éxito de ventas.

Ilustración 5 Super Nintendo versión mini

Las características de estos juegos son: gran dificultad, *permadeath*, juego orientado a puntuación, gráficos sencillos y jugabilidad adictiva.

2.2. Revisión de la tecnología empleada

Antaño, este tipo de juegos se programaba exclusivamente en el ensamblador de la máquina o placa arcade. Era evidente el coste días/persona que esto suponía; además de tener que contar con expertos en la materia.

Actualmente, se encuentra apropiado utilizar herramientas que faciliten tareas no imprescindibles al creador de videojuegos, hasta el punto de que incluso es posible crear este tipo de juegos sin programación alguna (con la evidente reducción de posibilidades que ello supone).

Nos encontramos con el uso de los entornos de desarrollo integrado ya comentados, como Unity3D o el escogido para el proyecto: Game Maker. También debemos mencionar aquí, Godot, un motor similar a Unity3D con licencia abierta.

Los usuarios con mayor conocimiento y necesidades de libertad técnica suelen emplear un lenguaje de programación cercano al sistema (C, C++, Java o Swift) para desarrollar videojuegos más exigentes. En este caso se emplean las mismas herramientas que aquellas empleadas para otro tipo de software, acompañadas de utilidades externas para la creación de sprites o texturas.

Las librerías típicas para este tipo de desarrollo son libsdl, sfml y algunas nuevas como raylib o Cocos2D. Además de librerías orientadas a juegos en dos dimensiones, también podemos encontrar desarrollos hechos en OpenGL, una librería, a priori, más cercana a juegos en 3D.

En este caso, **se ha optado por Game Maker**, por disponer de una licencia para producción en sistemas de escritorio (Linux, MacOS, Windows), además de facilitar, como hemos comentado, las tareas propias de programación y fomentar un desarrollo orientado a la producción de videojuegos.

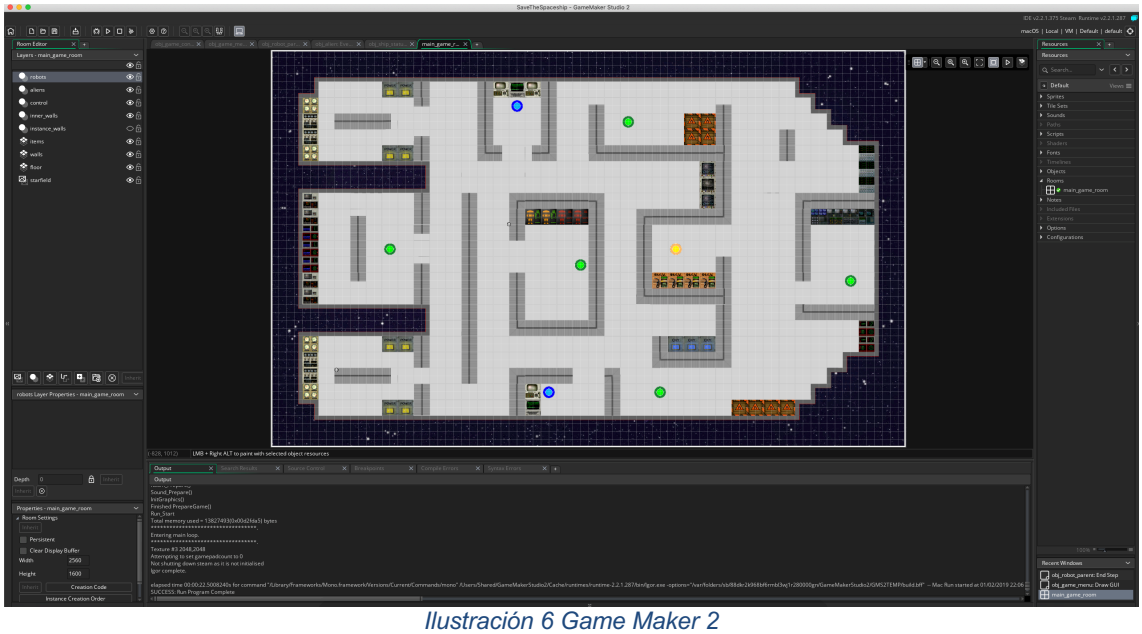


Ilustración 6 Game Maker 2

La alternativa más clara era Unity3D, pero sopesando esta tecnología frente a Game Maker suponía invertir en conocimiento para un lenguaje más complejo que GML, como es C# y una biblioteca aún más extensa. Además, Unity3D está orientado precisamente a juegos en 3D, siendo el 2D secundario; mientras que Game Maker sí que está pensado para 2D como elemento principal.

Las características principales de este tipo de software son:

- Editor de texto, depurador, compilador integrados.
- Editor de imágenes-sprites propio.
- Integración de sistemas de control de versiones de código.
- Biblioteca de métodos y clases orientada a videojuegos.

3. Definición del juego

3.1. Introducción

De forma abstracta, el juego presenta un reto de optimización de recursos al jugador. Este, es presentado al mando de un equipo de reparación robotizado cuya misión es que las naves espaciales que transportan mercancías de valor lleguen al lugar de destino.

Estas naves, sin tripulación humana, poseen un número limitado de efectivos robots para las tareas de mantenimiento, reparación de averías, impacto de

meteoritos e invasiones alienígenas. La misión del jugador es orquestar estos recursos para evitar que la nave quede inoperativa, varada en el espacio o, sencillamente, se destruya por los daños recibidos.

Durante el viaje de regreso a la base sucederán todo tipo de eventos relacionados con averías y desastres que requerirán que el jugador deba gestionar eficazmente los recursos de los que dispone para que la nave mantenga dentro de lo posible su integridad, así como la de los propios robots.

3.2. Referencias a videojuegos existentes

La referencia más cercana al juego en el aspecto gráfico es, posiblemente, el videojuego Faster Than Light. En este juego, el jugador posee el control de la tripulación de una nave espacial que ha de huir del enemigo por una galaxia y enfrentarse a naves piratas, misiones de auxilio o comerciales.

En Faster Than Light, el jugador puede mover a la tripulación a lo largo de la nave para gestionar las averías que se producen como consecuencia del combate con naves enemigas.

No es la intención, desde luego, crear un clon de FTL. Sin embargo, si bebe de su inspiración original: comandar una nave en situación de crisis, donde se deben gestionar unos recursos limitados. No obstante, mientras que FTL es un juego de estrategia por turnos, “Save the Spaceship!” Es un arcade puro, en el que el tiempo corre sin detenerse y requiere del jugador una atención y reacción plena.

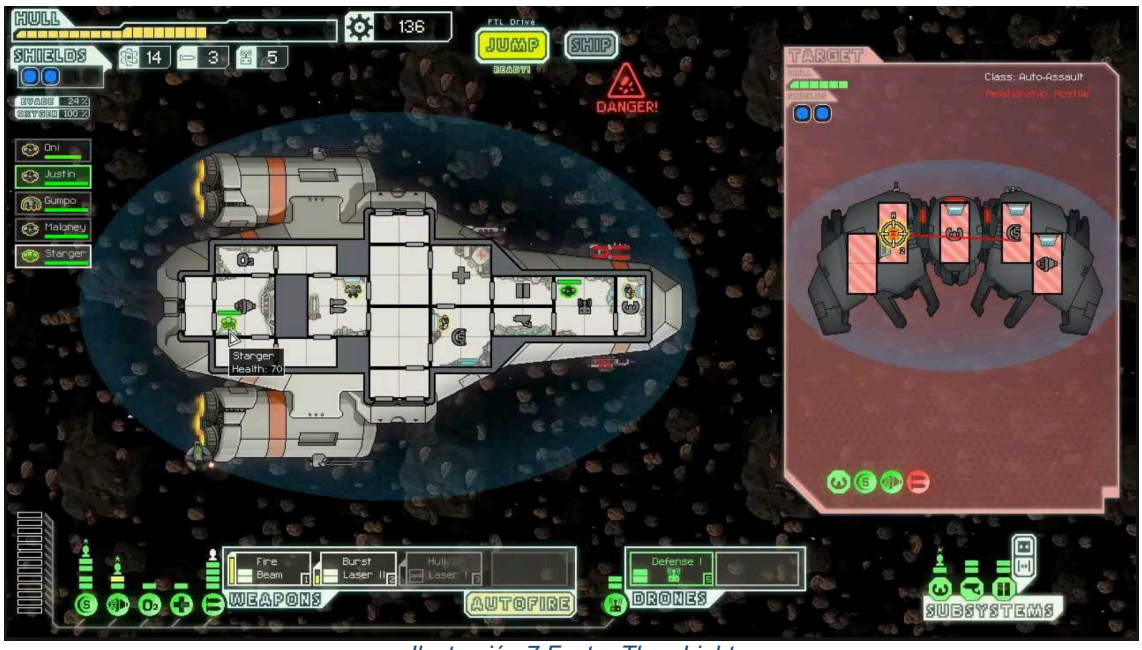


Ilustración 7 Faster Than Light

3.3. Tipo de interacción juego-jugador

La interacción en juego se realizará completamente mediante ratón y teclado (accesos directos). Esencialmente, el ratón se utiliza para mover un cursor mediante el cual, el jugador podrá, previamente a seleccionar un robot, indicar hacia donde debe dirigirse y qué debe reparar. El teclado podrá usarse como acceso directo para elegir el tipo de robot.

El juego se realizará en un entorno clásico de 2D con perspectiva cenital. Presenta un plano de la nave, donde se podrá observar su interior, los componentes de esta y los robots moviéndose a través de ella, además de efectos visuales y personales que representan las averías o alienígenas invadiendo la nave.

3.4. Plataforma de destino

Las plataformas de destino son las computadoras de escritorio. En concreto, sistemas GNU/Linux, Apple MacOS y Microsoft Windows. Esta decisión obedece meramente a cuestiones de licenciamiento del motor de videojuegos escogido.

4. Conceptualización

4.1. Historia, ambientación y trama

En el futuro, los viajes espaciales entre colonias terrestres son habituales. El intercambio comercial no conoce fronteras y miles de naves espaciales autopilotadas cruzan la galaxia para comerciar con bienes materiales. Pero existe un problema; un grave problema. Al no poseer tripulación humana, las naves requieren de unidades robotizadas para reparar las numerosas averías que pueden suceder durante estos vuelos.

Empresas especializadas en este tipo de mantenimiento sideral son contratadas con la finalidad de gestionar de forma remota los robots de mantenimiento. No es suficiente con la inteligencia artificial, ya que no hemos avanzado en esta materia mucho más allá del reconocimiento de imágenes, manipulación de objetos y movimientos sencillos. Es necesario que un humano supervise a los robots, gestiones sus tareas y active o desactive características de la nave para garantizar la integridad de esta.

4.2. Definición de los personajes y elementos principales

El operador

Es el jugador. No posee una representación física en el juego. Es un operador de la compañía contratada para dirigir las reparaciones de la nave; lo hace de forma remota a través de una terminal.

Los robots

Son los elementos principales para reparar las averías de la nave. Poseen una vida limitada, por ejemplo, si estos intentan extinguir las llamas, sufrirán daño al hacerlo. Los robots pueden repararse entre sí y recargar la batería de un robot descargado.

El fuego

Son las contingencias que harán que la nave vaya perdiendo integridad. Se producen de manera eventual. La dificultad radica en que estas pueden expandirse, afectar a otras partes de la nave o provocar daño en los robots al intentar estos repararlas.

La nave

Las naves pertenecen a las compañías de comercio. Son naves de transporte no tripuladas por humanos. Dentro pueden llevar minerales, créditos, material radioactivo, etc. Para viajar a velocidades más allá de la luz, poseen un motor factorial propulsado por un núcleo. Uno de los peligros es que este núcleo se quede sin combustible, dejando la nave varada.

Los alienígenas

Son seres con capacidad de tele-transportación que vagan por el espacio y se alimentan con el combustible de la nave. Si detectan la nave, aparecerán dentro y consumirán el combustible de la nave con el siguiente peligro. Los robots pueden neutralizar esta amenaza cazándolos.

4.3. Interacción entre los actores del juego

Una vez comienza la partida, el jugador observará la nave desde un plano cenital, con su interior descubierto a modo de plano de estancias. También podrá ver los indicadores vitales de la nave: estado del casco, combustible y progresión del viaje.

Durante el viaje de la nave, se sucederán las averías, impactos de meteoritos e invasiones de alienígenas. Por ejemplo, el jugador verá como una instancia de la nave comienza a arder por un cortocircuito. Entonces se activará una alerta

de avería y se deberá seleccionar al robot más cercano o apropiado para la tarea.

Una vez se selecciona al robot o robots, estos se dirigirán donde apunte el operador (jugador) y comenzarán a reparar la avería o detener al alienígena. Es posible que los robots se dañen durante las reparaciones o el trayecto hasta las estancias donde se encuentra la avería, por lo que o bien deberá repararlos con otros robots o llevarlos de vuelta a la estación de robots para que sean reparados.

4.4. Objetivos planteados al jugador

Como se ha expuesto en los apartados anteriores, el jugador debe coordinar los robots para que neutralicen todas las amenazas y la nave llegue a buen puerto con suficiente combustible e integridad del casco.

También deberá velar por los propios robots, ya que si estos consumen su batería o su integridad se ve debilitada no podrán moverse y deberán ser reparados. Estas reparaciones o recargas pueden efectuarse en una estación habilitada para ello. Otros robots también pueden reparar y recargar la batería de unidades caídas (a expensas de su propia batería)

Una vez la nave llegue a la Tierra, finalizará el nivel y comenzará el siguiente que verá incrementada su dificultad.

El resultado y objetivo final es que el jugador incremente su capacidad de atención y evaluación de amenazas con la dificultad que conlleva realizar estas acciones a contrarreloj y con recursos limitados.

4.5. Arte conceptual

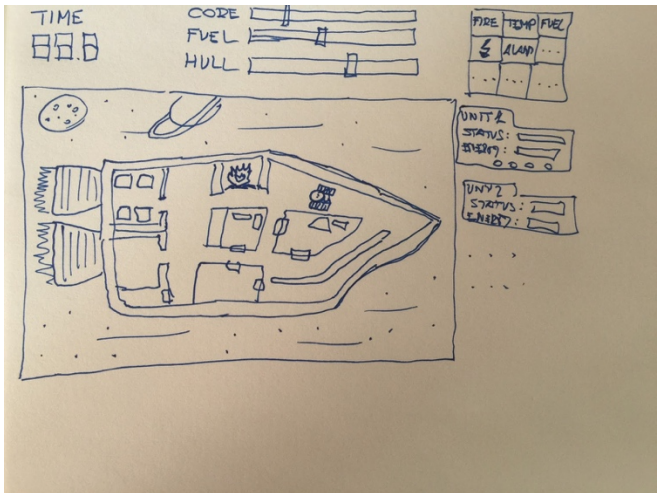


Ilustración 8 Boceto inicial del juego

En la imagen de arriba podemos ver un boceto inicial, presentado cuando se creó el documento de diseño del videojuego. Este diseño fue progresando hacia la forma final, donde toma más forma de arcade y su estilo de juego se vuelve más orientado a producir respuestas rápidas del jugador.

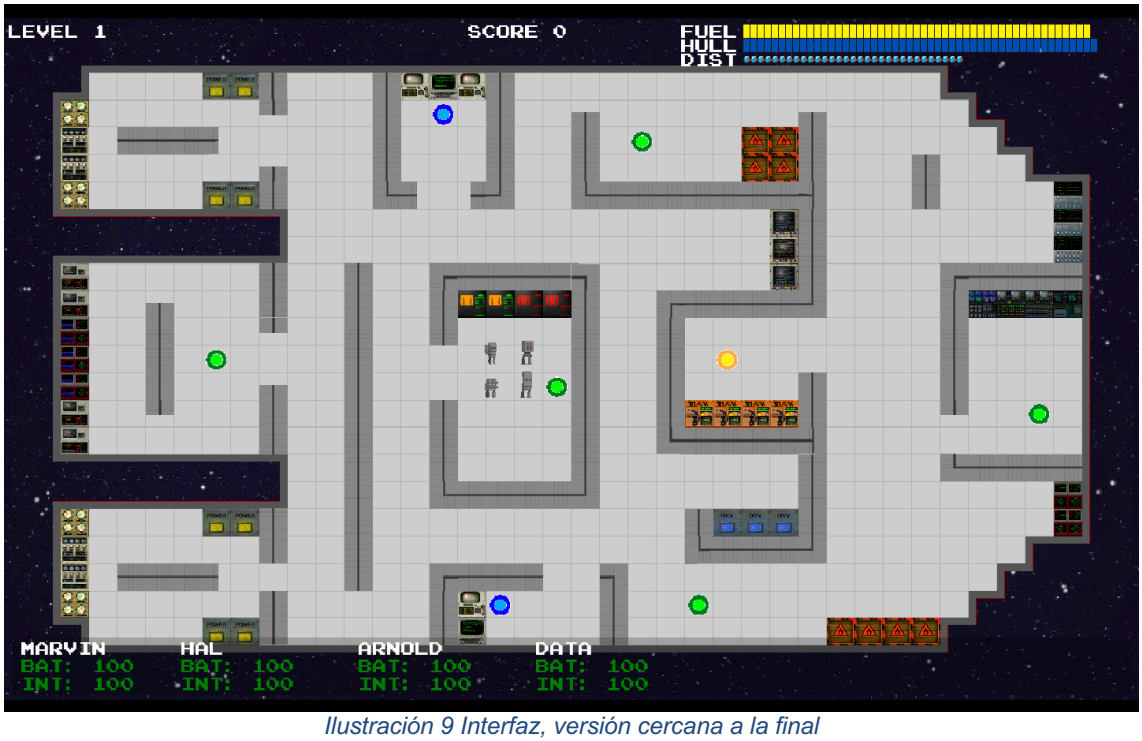


Ilustración 9 Interfaz, versión cercana a la final

Finalmente, podemos ver como se reduce el nivel de instrumentación mientras que se amplía la zona de juego. Durante el desarrollo del videojuego se pivota más hacia a un arcade que la idea original, con mayor componente estratégico.

5. Diseño técnico

5.1. Elección final del entorno de desarrollo, justificación e inventario.

En cuanto a videojuegos, existen dos aproximaciones fundamentales: o bien construirlos mediante la opción lenguaje de programación + librería o a través de un entorno integrado de desarrollo.

La primera de las opciones: “lenguaje + una librería adecuada”, nos permite reutilizar nuestro conocimiento respecto al lenguaje elegido (si lo dominamos). Pero posee la dificultad adicional de tener que reducir el nivel de abstracción en la implementación de los mecanismos del juego al nivel que posea la librería elegida. Por ejemplo, la librería conocida como SDL (Simple DirectMedia Library) nos permite usarla mediante una gran cantidad de lenguajes de

programación, pero no posee rutinas de alto nivel para, por ejemplo, hacer que un personaje trace una ruta hacia una habitación.

La segunda opción nos deja en entornos como Unity3d y GameMaker o similares. Perdemos cierta libertad en la implementación a bajo nivel de mecanismos de juego, pero a cambio ganamos en tiempo de desarrollo, ya que son entornos de desarrollo orientados, precisamente, a la creación de videojuegos.

Se descarta la primera opción, básicamente porque el tiempo disponible de desarrollo es muy limitado. Implementar las rutinas propias de un videojuego (incluso si nos decantamos por una librería o grupo de librerías que ya dispongan de ellas) es costoso en tiempo.

Una vez se decide evaluar los entornos integrados de desarrollo, se valoran cara a cara los ya mencionados en el párrafo anterior. Finalmente, se decide implementar el videojuego con GameMaker, por las siguientes razones:

Como se ha dicho anteriormente en otro apartado de esta memoria, se dispone de una licencia para trabajar plenamente con dicho entorno. Además, el entorno escogido está orientado principalmente a juegos en 2D (aunque Unity3D soporta 2D es una característica secundaria) Es más sencillo que Unity3d en cuanto a opciones disponibles, lo cual disminuye la curva de aprendizaje necesaria para abordar su uso.

No obstante, merece la pena comentar que el alumno autor de esta memoria no posee conocimiento con profundidad de estos entornos, tan solo el necesario para evaluar su ajuste a las necesidades del proyecto. En realidad, se dispone de un conocimiento profundo de C y C++, pero no de las librerías típicas para la creación desde cero de un videojuego con estos lenguajes.

Por todo ello, barajando todas las opciones posibles, **Game Maker resultó ser la opción más adecuada en relación con el tiempo disponible y las prioridades a cumplir**.

Para la **creación de sprites** se recurrió al programa Aseprite, del cual se disponía ya de una licencia de uso (aunque es un software con licencia libre) y cierta experiencia. Aseprite es un software diseñado expresamente para la creación de sprites en 2D.

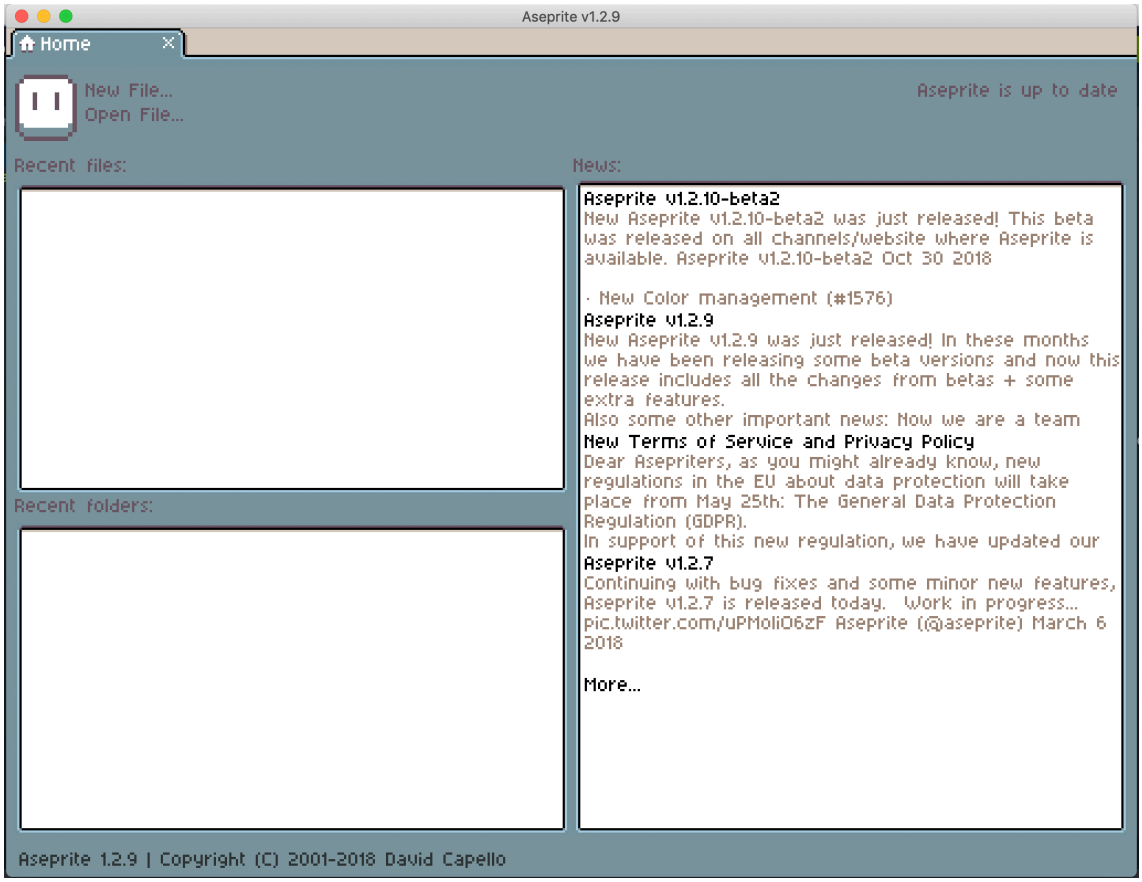


Ilustración 10 Detalle de la pantalla inicial de Aseprite

5.2. Requerimientos técnicos del entorno de desarrollo

Es necesario poseer un ordenador con sistema operativo Linux, MacOS o Microsoft Windows para ambos programas (Game Maker y Aseprite). Los requerimientos técnicos de estos son los siguientes:

Game Maker 2 (de la página del fabricante):

*"The minimum system **requirements** needed to run**GameMaker** Studio 2 are: 64bit Intel compatible Dual Core CPU, 2GB RAM, DX11 compliant graphics card, Microsoft 64bit Windows 7 (or above), HDD with at least 3GB free disk space"*

Aseprite (de la página en Steam del fabricante):

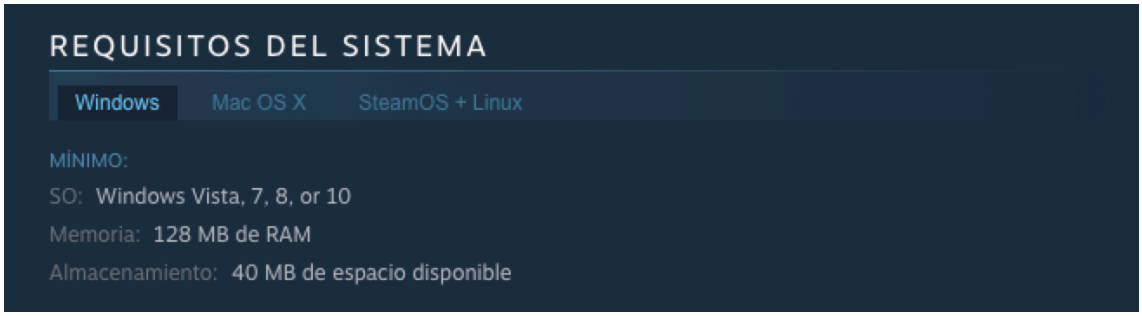


Ilustración 11 Requisitos del sistema para Aseprite

5.3. Inventario de recursos empleados (assets)

A continuación, se listan los recursos gráficos y sonoros empleados para el videojuego. Tanto los creados para el videojuego como aquellos que se han reutilizado de fuentes adecuadas y siempre velando que la licencia que disponen sus creadores permita tal uso.

5.3.1 Recursos propios, creados para el videojuego

Tileset de creación propia para las paredes exteriores de la nave:

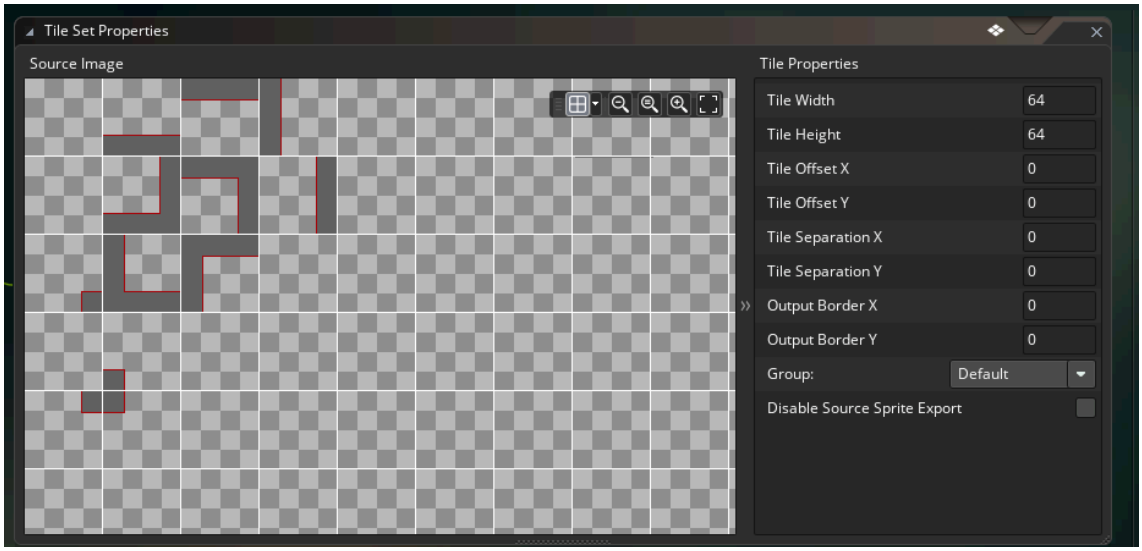


Ilustración 12 Tileset empleado para las paredes exteriores

Para las paredes interiores se usan sprites de producción propia:

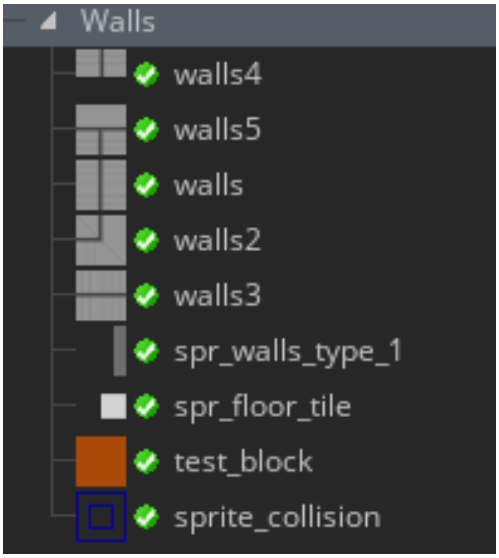


Ilustración 13 Sprites diversos para interiores y marcadores de prueba

El marcador de niveles de combustible, integridad del casco y tiempo de llegada a la Tierra, también es de producción propia. El sprite de la Tierra, bastante elaborado, tuvo que ser reducido ya que con el tamaño que se observa en la ilustración era desproporcionado. Lamentablemente, el tamaño actual esconde muchos de los detalles que el tamaño original muestra.

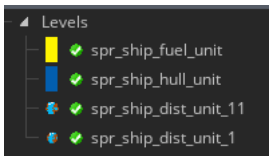


Ilustración 14 Detalle de los sprites para marcadores

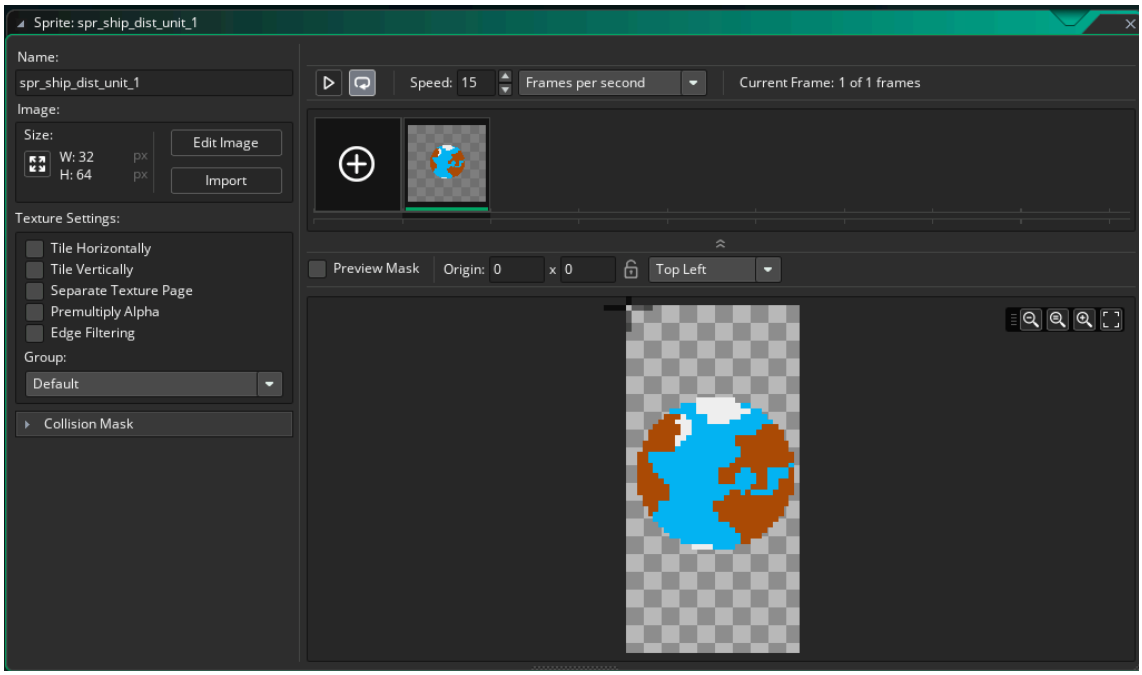


Ilustración 15 Detalle del sprite que simboliza el marcador de llegada a Tierra

Resto de items del juego, estaciones de recarga, conversión de combustible, aliens y reparación del casco. Estos recursos son sencillos adrede. Se quiso disponer de una misma imagen conceptual para señalar donde deben ir los robots para disparar ciertas acciones de reparación o recarga.

El fuego y el alien si poseen sus propias animaciones para dotar de dinamismo y llamar la atención del jugador.

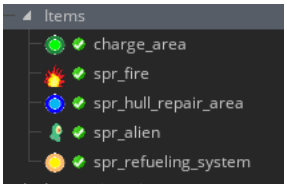


Ilustración 16 Detalle de items del juego

Animación creada para el movimiento del alien:

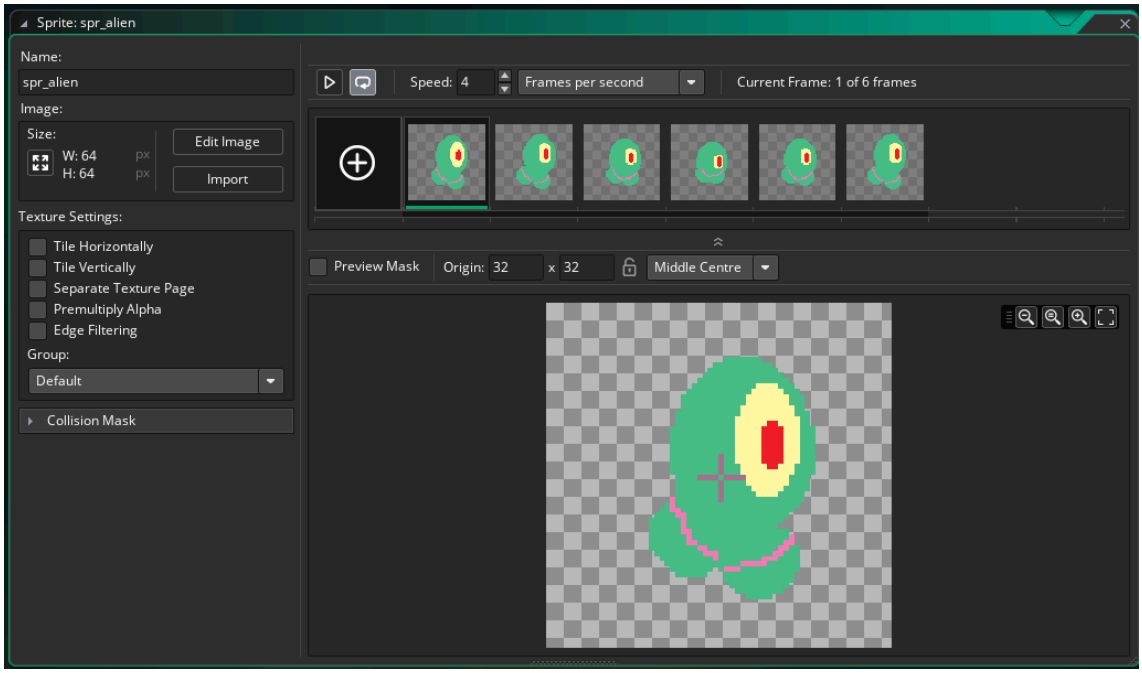


Ilustración 17 Animación del alien

Esta animación se creo en Aseprite y se importó a Game Maker. No está basada en ningún arte previo, aunque es fácil ver que el alienígena procede del imaginario común que los diferentes comics y películas de animación han marcado: piel verde, ojo único y desproporcionado.

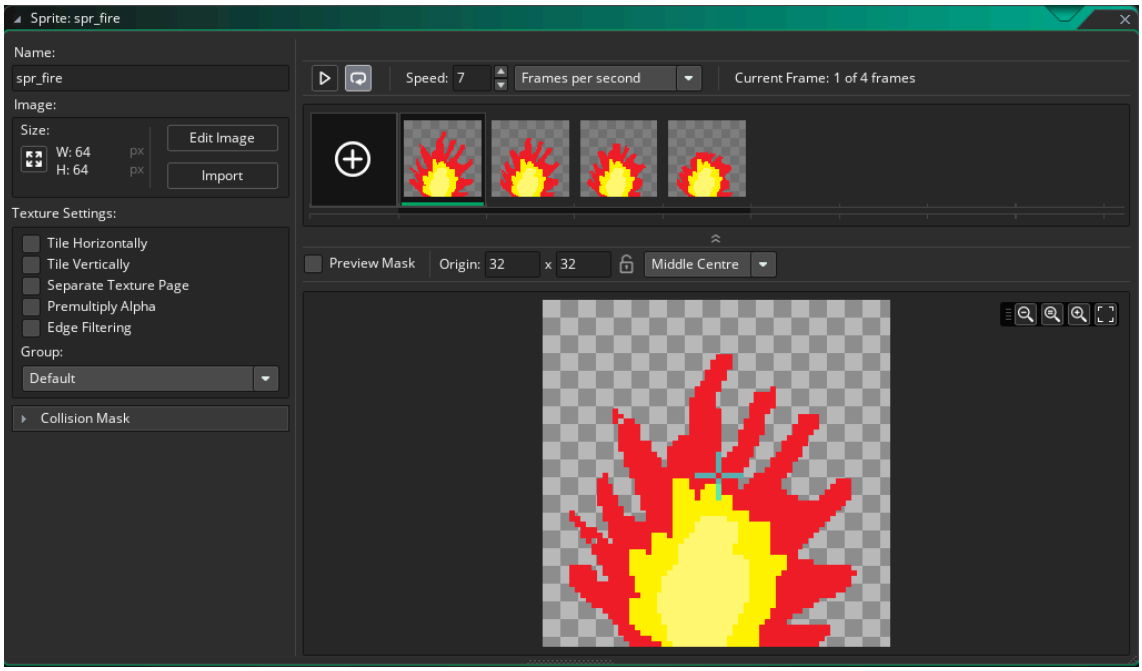


Ilustración 18 Animación del fuego

La animación del fuego es simple, se trata del mismo frame con las llamas progresivamente recortadas. Es sencillo, pero funciona bien sobre el terreno de juego.

5.3.2 Recursos reutilizados

Se ha empleado un *tileset* con licencia libre, procedente de OpenGameArt, para las decoraciones de interiores de la nave y el suelo.

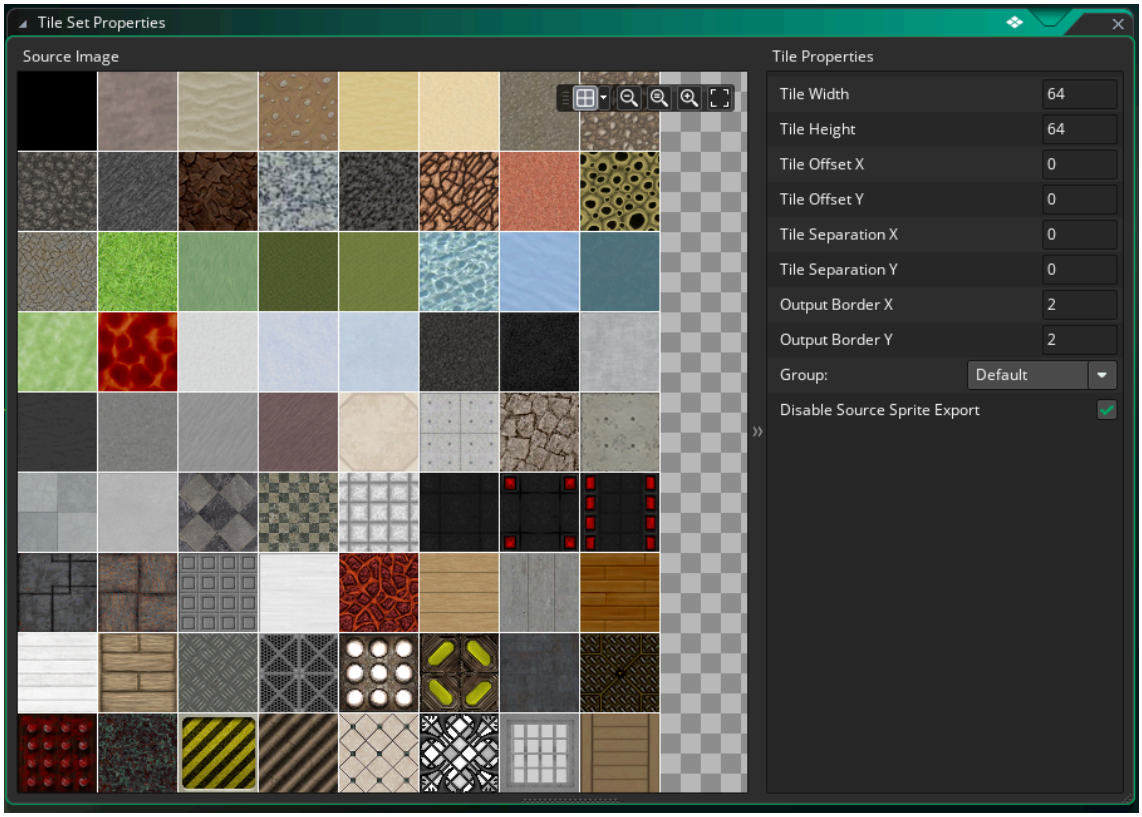


Ilustración 19 tileset usado para decoraciones

Se ha empleado parte del siguiente tileset de licencia libre para los robots, procedente de https://0x72.itch.io/16x16-robot-tileset

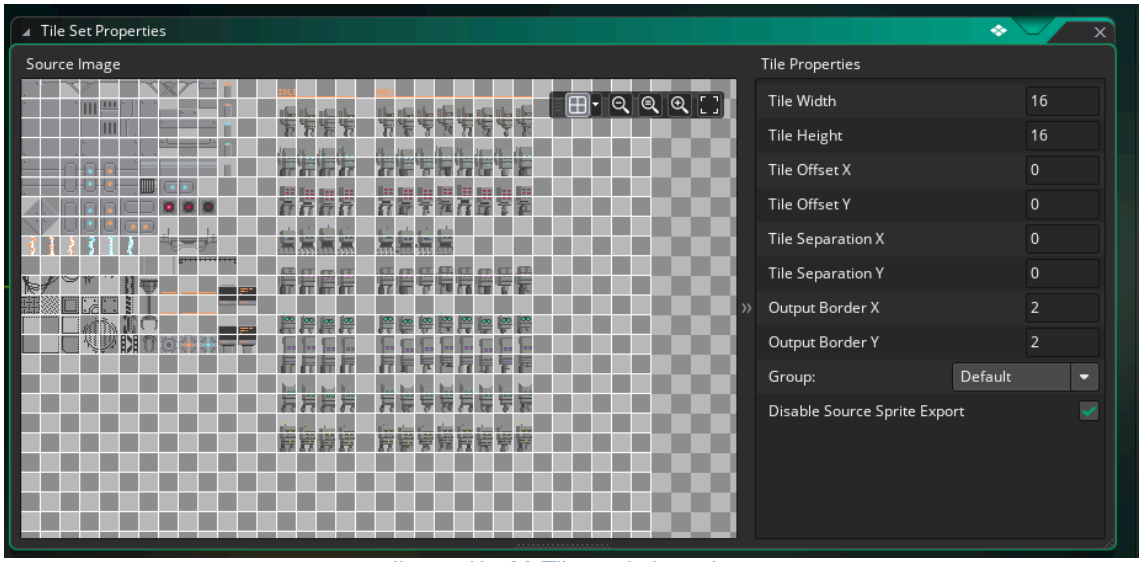


Ilustración 20 Tileset de los robots

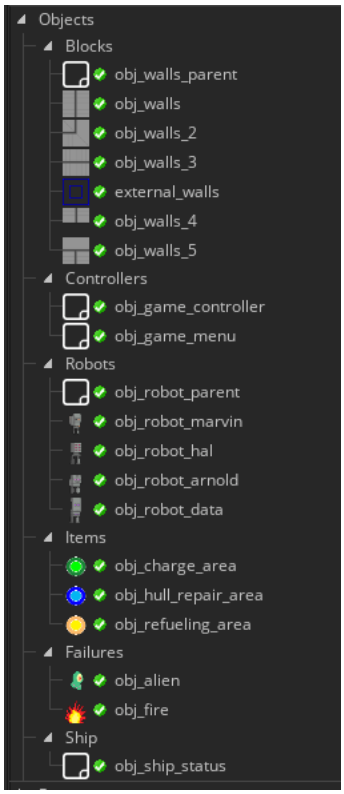
Todos los sonidos empleados proceden del banco de sonidos open source FreeSound.

Por ejemplo:

https://freesound.org/people/Romariogrande/sounds/396239/
https://freesound.org/people/broumbroum/sounds/50561/
https://freesound.org/people/Supakid13/sounds/210018/
https://freesound.org/people/AudioPapkin/sounds/430977/

5.4. Esquema arquitectural del videojuego

Estos son todos los componentes del videojuego:



- Los bloques son objetos sobre los que ninguna entidad del videojuego podrá o deberá atravesar. Son objetos sólidos, desprovistos de animación que marcan los límites físicos de desplazamiento.

- Hay dos controladores principales, el de juego que inicia todas las instancias y controla el ciclo de juego y un controlador especial para el menú de opciones y pausa. Cuando el usuario inicia el juego o lo pausa, el controlador de juego cede el control al de menú, para que este pueda presentar una lista de opciones. De este modo se separa la acción de la jerarquía de páginas típicas de un menú.

- Los robots son las entidades principales que maneja el jugador. Cada uno de los cuatro robots que existen en el juego posee sus propias características, pero existe una clase común: obj_robot_parent que asume aquellas que les son comunes a todos ellos. Aunque el concepto pueda ser parecido al de la herencia del paradigma orientado a objetos, no posee la potencia de este último.

- Los ítems, son objetos sobre los que los robots interactúan para reparar la nave, a ellos mismos o realizar recargas de batería y combustible. Su existencia es sencilla, no poseen elementos comunes ya que las colisiones se tratan desde los robots.

- Los fallos o 'failures' son dos con entidad, el fuego y los aliens. Al poseer comportamientos totalmente distintos, no poseen un objeto común. Los meteoritos no poseen parte gráfica y son instanciados y emulados desde

el objeto de control del juego; así como el efecto de colisión con uno de ellos.

- La nave posee un objeto a modo de placeholder que contiene un grupo de variables y métodos asociados a su estado. Por ejemplo, si un alien está consumiendo combustible, el control de juego sustraerá una cantidad determinada de su variable combustible.

El componente más complejo con diferencia es el controlador de juego, por supuesto. Esto es debido a las múltiples comprobaciones de reglas, dificultad, control de instancias, creación de rutas, aceptación del input del jugador, etc.

Breve anotación respecto a la IA

No existe IA en el juego como tal, a reseñar el comportamiento de los aliens, que, aunque sus rutas sean aleatorias, vienen marcadas por un pequeño fragmento de código que tiene en cuenta las posiciones de los robots para evitar encontrarlos por el camino; en otras palabras, evitan a los robots.

```
// If alien is not moving then it needs another path
if (x == xprevious and y == yprevious) {
    randomize();
    _x = irandom_range(-target_offset, target_offset);
    _y = irandom_range(-target_offset, target_offset);
    if (mp_grid_path(alien_grid, alien_path, x, y, x+_x, y+_y, false)) {
        path_start(alien_path, 2, path_action_stop, 0);
    }
}
```

Ilustración 21 Detalle del código que evita rutas con robots en la línea de visión

5.5. Diseño de niveles

El juego está diseñado para progresar sobre distintos niveles, no obstante, no varía el diseño del escenario. Dotar al juego de esa modificación hubiera complicado notablemente el código, ya que debía incluir el calculo de bloques sólidos, implicado en la creación de rutas y colisiones. No obstante, se discute una vía para la futura incorporación de un sistema procedimental de creación de interiores.

El juego, una vez comenzada la partida posee el siguiente aspecto (Se aportan anotaciones):

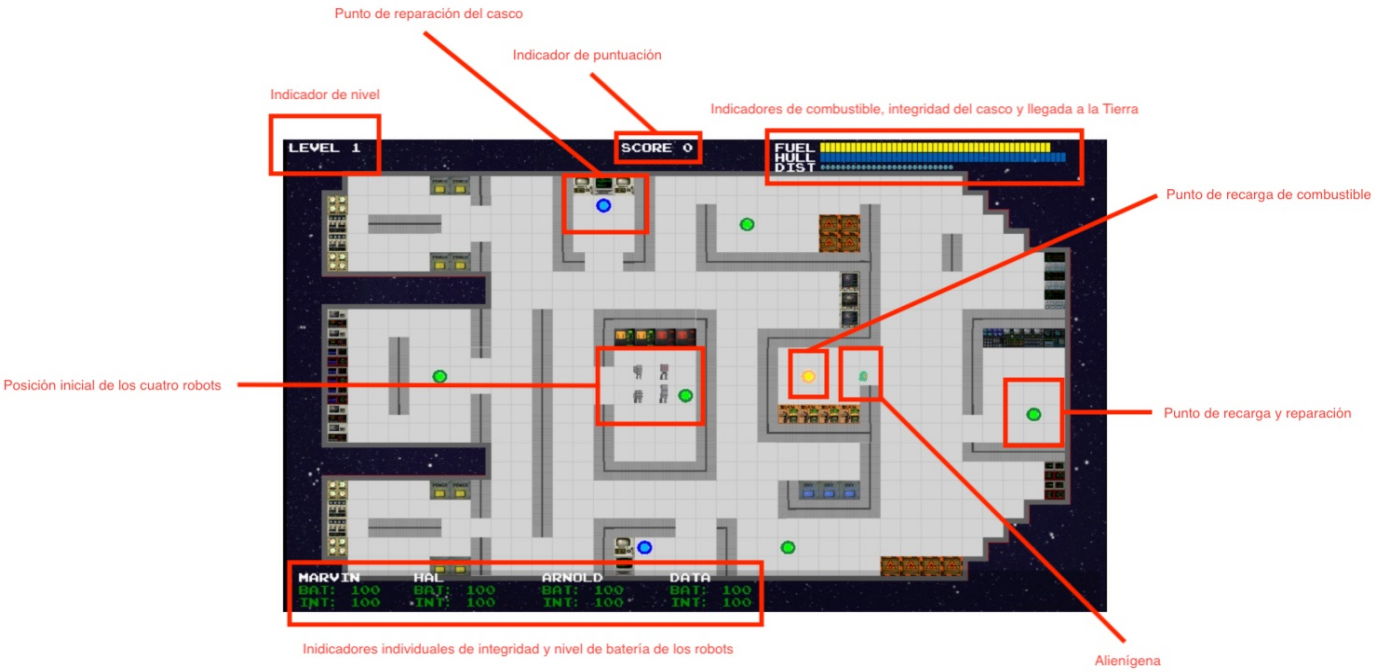


Ilustración 22 Esquema explicativo de niveles

En cada nivel sucederán eventos tales como incendios, invasiones de alienígenas e impactos de meteoritos. Todo ello, hará que la integridad del caso de la nave y los propios robots se deteriore. Los alienígenas, junto con el consumo normal de combustible, hará peligrar el avance de la nave.

La misión, por tanto, de los robots, es proteger la nave, su combustible y a ellos mismos. Depende de la pericia del operador (el jugador) la coordinación de estos para llegar a buen fin.

5.6. Discusión de la dificultad y su cálculo

En primer lugar, la dificultad es incremental. Es decir, cuando subimos de nivel, incrementa, lógicamente, la dificultad.

El núcleo de la idea está en la cantidad de eventos que se producen. Cada segundo se efectúa un “sorteo”:

```
// First, let's see if we can trigger an event
if global.event_ticker > 60 {
    randomize();

    // The greater the level the more chances we have
    var upper_range = clamp(30 - global.level, 1, 30);
```

Ilustración 23 Dificultad 1/3

El nivel determina la cota superior hasta un límite de 30 (no se espera que nadie con capacidades humanas normales llegue a ese nivel), a partir de ahí la cifra sigue siendo la misma, pero si o si, se producirá uno o varios eventos por segundo sin posibilidad alguna de que “no ocurra nada”.

Por ejemplo, en la creación de alienígenas:

```
// Get some tickets for aliens
for (var i = 0; i <= global.level ; i++) {
    var ticket = irandom_range(1, upper_range);
    var winner = round(upper_range / 2);
    var spawn_cell = irandom_range(0, array_height_2d(alien_array) - 1);
    if (ticket == winner and (global.number_of_concurrent_aliens <= global.level)) {
        // Spawn an Alien
        var alien = instance_create_layer(alien_array[spawn_cell, 0], alien_array[spawn_cell, 1], layer_get_id("control"), obj_alien);
        ds_map_add(global.alien_map, alien, alien);
        continue;
    }
}
```

Ilustración 24 Dificultad 2/3

El sistema hace el “sorteo” *n* veces. Cuanto mayor sea el nivel menor será el rango de enteros y mayor el número de sorteos. Cuando “toca”, se procede a crear un evento en una posición determina de forma aleatoria.

Para no crear una situación de caos absoluto en los primeros niveles se posee un “seguro” que bloquea el número de eventos concurrentes. Esto permite dilatar un poco más la dificultad (ya de por sí elevada). Por ejemplo, es notable este uso cuando el fuego trata de extenderse:

```
for (var i = 0; i <= (global.level / 2) ; i++) {
    var ticket = irandom_range(1, upper_range);
    var winner = round(upper_range / 2);

    var spawn_cell = irandom_range(0, array_height_2d(fire_array) - 1);

    if (ticket == winner and (ds_list_size(global.fire_list) < global.level)) {
        // Spawn an fire event
        var fire = instance_create_layer(fire_array[spawn_cell, 0], fire_array[spawn_cell, 1], layer_get_id("control"), obj_fire);
        audio_play_sound(snd_fire_alarm, 10, false);
        ds_list_add(global.fire_list, fire);
        continue;
    }
}
```

Ilustración 25 Dificultad 3/3

Se mantiene una lista de “focos activos”, si aun no se han extinguido un número concreto, relacionado con el nivel actual, no se extenderá aun más. Dejando algo de respiro al jugador.

5.7. Sistema de partículas

Una de las mejoras incluidas a raíz de los comentarios de los profesores consultores en la PEC-3 fue la inclusión de sistemas de emisión de partículas para crear ciertos efectos cuando el jugador efectuaba ciertas acciones.

Esto ha resultado en un incremento espectacular de los efectos del juego y el feedback visual de que posee el jugador.

Se han creado dos efectos con partículas: cuando se extingue el fuego (se crea una humareda típica) y cuando se da caza a un alien (se crea un efecto de desintegración).

El código a tal efecto es similar para ambos emisores:

```
// Particle
part_type = part_type_create();
part_type_shape(part_type, pt_shape_smoke);
part_type_alpha3(part_type, 0, 1, 0);
part_type_size(part_type, 0.2, 0.5, -0.005, 0);
part_type_colour_mix(part_type, c_gray, c_black);
part_type_speed(part_type, 0.1, 0.5, 0, 0);
part_type_direction(part_type, 0, 359, 0, 0);
part_type_gravity(part_type, 0.02, 90);
part_type_life(part_type, room_speed * 2, room_speed * 4);
//part_type_blend(part_type, true);

// Particle emitter
part_emitter = part_emitter_create(global.part_system);
part_emitter_region(global.part_system, part_emitter, x, x+16, y, y+16, ps_shape_ellipse, ps_distr_gaussian);
part_emitter_stream(global.part_system, part_emitter, part_type, 2);

// Lifespan
alarm[0] = 60;
```

Ilustración 26 Detalle código emisión partículas

Existe un sistema global de partículas sobre el que se crean emisores en el punto que queramos (típicamente, donde se establece la acción).

6. Informes de experiencia de usuario

Tal y como se expone en la “Guía de estudio”, se indicaba opcionalmente la inclusión de comentarios de usuarios. Se pensó que sería buena idea incluir una etapa en la que un grupo reducido de usuarios pudiese probar el juego y comentar los aspectos tanto positivos como negativos.

Este tipo de información resultó ser enormemente útil durante el desarrollo. **Sin duda, ahora se entiende el interés y la importancia de los desarrolladores en incluir un programa de acceso a la alfa o beta para recabar críticas de los usuarios**.

A continuación, se relatan las distintas experiencias:

Ignacio, 36 años, Ingeniero Informático.

Ignacio es un jugador experimentado. Su ayuda ha sido sobre todo captar fallos técnicos, incluso con informes por cada versión que le proporcionaba, por ejemplo:

“STS version 43455

Incidencias:
-Cuando arranca a resolucion baja no se ven las opcione de instrucciones y salir, activar pantalla completa desde el inicio, o ponerlas en otra posicion del menu inicio
-En las instrucciones pone que la navegacion con la ayuda es A/S, pero es con A/D.
-Los enemigos no quitan vida a los robots, los robots son indestructibles.
-No se puede ir mas alla del nivel 5, las gasolina se acaba aunque lleve desde el primer momento del nivel 1 a un robot alli.
-La colision entre los recargadores y los robots es muy pequeña, hay que colocarse en un punto muy concreto o no recarga.”

“STS version 43456
Incidencias:
-No arranca de inicio. Tampoco responde al F4.
-Como consecuencia el sistema de coordenadas queda tocado al no ponerse en full screen. Los robots no van a donde pulso.
-No puedo probar mas porque eso me limita la experiencia.”

Marcos, 39 años, Licenciado en Empresariales.

Interesante. Divertido. “Está chulo”. No ve bien ni le queda claro cual es el robot seleccionado (del mismo modo que los profesores consultores aconsejaron en la evaluación de la PEC-3). Este punto se corregirá para la entrega final. El juego le parece muy bueno, desafiante.

Echa de menos que al cambiar de nivel no cambie la nave. Esto se ha discutido más arriba y es tema de cambios futuros en líneas de trabajo adicional y extensión del juego más allá de la vida del trabajo fin de grado.

Ainoa, 13 años, Estudiante.

Le parece divertido, pero es difícil. Cuando no pasa del nivel 3 o 4 se frustra bastante y no vuelve a el. Le parecen complicados los controles para seleccionar los robots. Dice que sería mejor pulsar sobre ellos con el ratón y luego llevarlos a otro lugar.

Laura, 30 años, Social Media.

Le encanta el aire retro. No es su estilo de juego, pero lo encuentra curioso y algo divertido. Dice que llega un momento que hay tantas cosas en pantalla que no sabe que hacer y llega a ser desesperante. También concluye que no se sabe muy bien que robot está seleccionado en ese momento.

7. Conclusiones

7.1. ¿Qué hemos aprendido?

El conocimiento principal ha sido pasar por la experiencia de plantearse unos objetivos, planificar unos hitos, detectar desviaciones de la planificación, corregir las expectativas demasiado ambiciosas, agregar nuevos aspectos sin modificar sustancialmente la programación y finalizar un proyecto en el cual se ha disfrutado enormemente.

En el aspecto de la gestión de proyectos:

- Plantear objetivos
- Trazar un plan cronológico y de administración de recursos para llegar a aquellos.
- Detectar desviaciones y corregirlas.
- Detectar oportunidades y agregarlas.
- Ser en partes iguales tanto ambiciosos como humildes en nuestras metas.
- Realizar presentaciones de resultados de forma progresiva.
- Recabar e integrar feedback de usuario.
- Finalización de un proyecto.

En el aspecto técnico de la creación de videojuegos:

- Usar un entorno de creación de videojuegos 2D
- Aprender el lenguaje de programación GML
- Técnicas básicas de creación y animación de sprites
- Programación de eventos-respuesta
- Programación de reglas de juego
- Detección de colisiones
- Creación de rutas
- Patrones de programación en videojuegos
- Comportamiento aleatorio de enemigos
- Diseño básico de niveles de juego
- Implementación de un ciclo de juego
- Ajuste de la dificultad de juego
- Efectos de pantalla

7.2. ¿Se han conseguido los objetivos planteados?

Los objetivos se han satisfecho en gran medida. El videojuego en su vertiente básica cumple con lo prometido. **No obstante, hay margen para su mejora y se deja la puerta abierta a implementarlas y ampliarlas en un futuro**. El videojuego se siente completo, jugable y en cierta medida adictivo según las experiencias ofrecidas por un grupo de usuarios.

A nivel personal del autor, el proyecto también es un "golpe de realidad" certero que nos muestra que hay tras las bambalinas. Muchas horas, muchas ilusiones y un tremendo esfuerzo. Sobre todo, si cabe, en el panorama *indie*, en los que los recursos y apoyos son tan necesarios como escasos. Valorar eso y conseguir la "vara correcta para medir" es un bien de incalculable estima.

7.3. Análisis del seguimiento de la planificación y metodología

El balance es positivo. No obstante, **existen varios aspectos que estaban incluidos en el documento de diseño del juego y no han llegado a implementarse**. Por ejemplo, el sistema de mejoras de los robots. Después de una fase, el jugador podría comprar mejoras para sus robots y acoplar esas decisiones a su estilo de juego, por ejemplo, añadir un módulo de velocidad, carga extra de batería, reparaciones, etc.

El motivo para no realizarlas no es otro que la magnitud de tiempo a emplear para integrarla en el juego. El diseño en papel ya advertía que no iba a ser fácil de implementar, ya que era preciso equilibrar las mejoras, balancear el sistema de puntuación y crear enemigos adicionales o tocar parámetros dinámicamente para que contrarrestasen esas mejoras y no terminasen por facilitar de manera desproporcionada el juego.

Del mismo modo, estaba previsto que las averías se propagasen por la nave, afectando a otras instancias. Esto también era complejo de solucionar por el mismo motivo comentado anteriormente. No obstante, se integró la aparición de "aliens" e impactos en el casco de la nave por parte de meteoritos, lo cual introduce variaciones en la temática, enriqueciendo la faceta de elementos adversos a combatir. Además, curiosamente ha proporcionado **una experiencia más cercana al estilo arcade** que se persigue en los objetivos iniciales.

Otro aspecto a tener en cuenta ha sido la **programación optimista del tiempo de aprendizaje** del entorno GM2, su lenguaje GML, la librería de funciones y

los patrones empleado. **Decididamente, el tiempo a invertir ha sido exponencialmente mayor al planeado**. Fue necesario trabajar más tiempo es estos aspectos para llegar a ser productivo con las herramientas empleadas.

La alternativa hubiera sido emplear herramientas ya conocidas por el autor, a saber: C o C++ y una librería ligera multimedia. No obstante, aun conocidos los lenguajes y librería, el tiempo de adaptación a los patrones de creación de videojuegos (ciclo de juego, entidades, colisiones, etc.) hubiese extendido del mismo modo el tiempo a invertir, con lo cual no se hubiera experimentado ganancia alguna.

7.4. Posibles líneas de trabajo adicional y extensión del proyecto.

Tal y como los profesores de la asignatura señalaron en la evaluación de la PEC-3 (entrega final), así como varios de los usuarios del juego en su versión preliminar, hubiese sido una buena característica la creación procedimental de las naves o al menos el interior de estas.

Actualmente, el juego posee una sola nave (escenario de juego). Esto condiciona bastante el apartado gráfico, ya que una vez el jugador aprende la estructura de la nave se vuelve repetitiva. Para evitar eso, un sistema procedimental es una línea excelente de investigación/inversión para mejorar ese aspecto del juego.

En principio, deberíamos tener en cuenta los elementos que componen el interior de la nave: paredes exteriores (en principio no variarán de posición), paredes interiores y puertas (simplemente vacíos donde no hay paredes) que permiten acceder al interior de estancias.

Las paredes interiores son las que nos definen las habitaciones, sus accesos y los pasillos que interconectan a estas. Deberíamos imponer un límite estructural inferior y superior a la dimensión de las habitaciones. Esto es debido a que no deberían ser ni muy pequeñas (en las que no cabría nada) ni desproporcionadamente grandes, ya que no dejarían espacio para otros elementos.

Del mismo modo, estas no podrían separarse ni mucho ni poco, para no dar la impresión artificial de incoherencia. Esto nos llevaría a crear un nivel de densidad de objetos en el escenario sumando la superficie de cada habitación creada. Si existen aun superficie suficiente y podemos introducir una nueva habitación se generan sus dimensiones y se superpone en el escenario, sino solapa con ninguna otra y puede colocarse, se añade.

Respecto a los pasillos debería ser sencillo generar simples filas de paredes interiores a modo de separadores, con la única condición de no limitar el paso

(dejar 1 casilla en ambas direcciones no ocupada) y dejar huecos, a modo de puertas, si los pasillos se alargan demasiado.

Aleatoriamente también y con suficiente separación, deberían colocarse otros elementos del juego, tales como las estaciones de recarga y reparación o el transformador de combustible.

Posteriormente, también se deberá adaptar la generación de eventos de fuego y *aliens* a la rejilla de juego libre, para que no colisionen en su aparición con objetos sólidos.

8. Glosario

arcade

Tipo de videojuego popular, nacido en los años 80 y 90 y relacionado con las máquinas recreativas.

feedback

En este contexto, el feedback de un usuario es el conjunto de opiniones, comentarios, críticas y experiencias respecto al uso del videojuego.

GM2 o Game Maker 2

Versión 2 del entorno de creación de videojuegos Game Maker.

GML o Game Maker Language

Lenguaje de programación orientado a eventos, dinámico e imperativo soportado por el entorno GM2.

Indie

Es un movimiento de creación de videojuegos (aunque el término aplica a otras corrientes artísticas) independiente de la industria tradicional y dominante. Se caracteriza fundamentalmente en la búsqueda de una expresión artística más que en un impacto en las ventas; en oposición a la industria dominante.

middleware

Software intermediario entre dos aplicaciones. En el ámbito del proyecto, es el entorno o librerías que facilita al programador la creación de un videojuego mediante la abstracción de técnicas o procedimientos encapsulados en clases o funciones.

Permadeath

La muerte del avatar del jugador representa el final de esa partida, haciendo que el jugador deba comenzar de nuevo al iniciar la siguiente.

Tileset

Conjunto de imágenes agrupadas para ser usadas mediante una referencia única en el juego. Una referencia contiene las coordenadas para buscar el “tile” o imagen adecuada dentro de un mismo conjunto de “tiles”. De esta forma nos permite ahorrar espacio al no tener que instanciar una misma imagen múltiples veces.

9. Bibliografía

1.- Game Programming Patterns, http://www.gameprogrammingpatterns.com/ (visitado en enero de 2019)

2.- Jamis Buck, Mazes for Programmers, The Pragmatic Bookshelf, 2015.

3.- Documentación online de Game Maker 2 https://docs2.yoyogames.com/, enero 2019.

4.- Mouse Dragging View (10/18)
https://www.youtube.com/watch?time_continue=38&v=R4hjsP6poqA

5.- Game Maker Studio 2: Tiles and Tilesets (10/18)
https://www.youtube.com/watch?v=hWxzZjVq-0w

6.- GameMaker Studio 2: Complete Platformer Tutorial (Part 6: Cameras & Tiles) (10/18)
https://www.youtube.com/watch?v=YbppozAV1Q8&list=PLPRT_JORnIupqWsjRpJZjG07N01Wsw_GJ&index=6

7.- 10 Rules for GM2 (10/18)
https://www.youtube.com/watch?v=G1WxKEk6Wrw

9.1. Ilustraciones externes usadas

Ilustración arcades
https://www.flickr.com/photos/goodrob13/17385639015

Ilustración Super Nintendo Mini
https://es.m.wikipedia.org/wiki/Archivo:Nintendo_Classic_Mini_Super_Nintendo_Entertainment_System.jpg

10. Anexos

10.1. Requerimientos e instalación

Es necesario disponer de un sistema operativo Microsoft Windows. Cualquier ordenador de sobremesa o portátil de menos de 10 años debería poder perfectamente ejecutar el videojuego.

No precisa instalación. Simplemente se descarga el siguiente archivo comprimido:

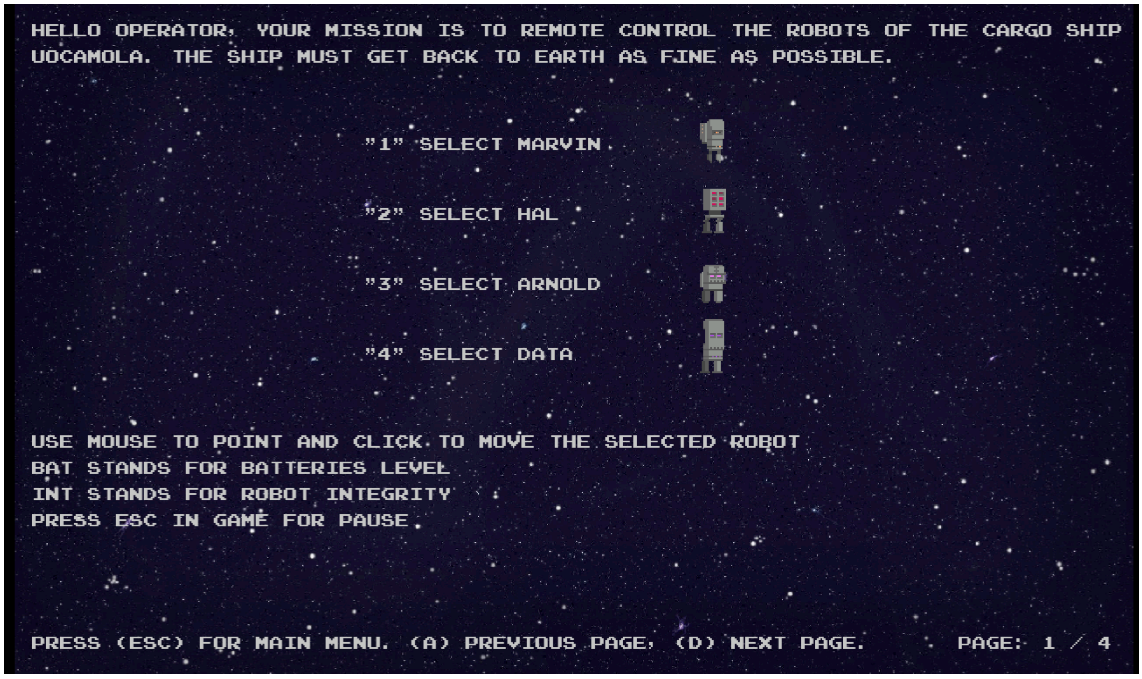
https://github.com/dgarnun/SaveTheSpaceship/raw/master/SaveTheSpaceship.zip

Se descomprime en cualquier lugar del disco y se ejecuta el archivo .exe

10.2.Manual de instrucciones

El videojuego alberga un apartado de instrucciones propio (en inglés) donde se describe la misión, los elementos, teclas y diversa funcionalidad.

En las diferentes páginas se puede observar el control de los robots, misión, funcionamiento de las estaciones de carga, etc.



10.3.Repositorio del código fuente

Tal y como se requiere, durante el proyecto se ha trabajado con el gestor de versiones de código git, y más concretamente con la aplicación web Github.

La url del repositorio es la siguiente:

https://github.com/dgarnun/SaveTheSpaceship

Existe una carpeta con el ejecutable final disponible en:

https://github.com/dgarnun/SaveTheSpaceship/tree/master/EJECUTABLE

Y, también existe un comprimido con el ejecutable en:

https://github.com/dgarnun/SaveTheSpaceship/blob/master/SaveTheSpaceship.zip